CUPRINS

6. *PL/SQL* – Subprograme

- Procedurile şi funcţiile *PL/SQL* sunt denumite subprograme *PL/SQL*.
- Subprogramele *PL/SQL*:
  - sunt blocuri *PL/SQL* cu nume;
  - au structura similară cu a blocurilor anonime:
    - secţiunea declarativă este opţională (cuvântul cheie *DECLARE* se înlocuieşte cu *IS* sau *AS*);
    - secţiunea executabilă este obligatorie;
    - secţiunea de tratare a excepţiilor este opţională.

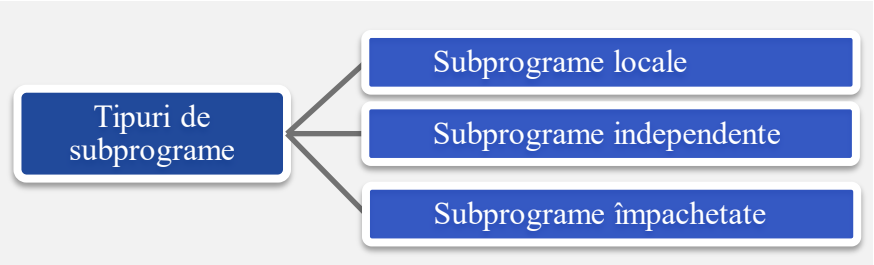

Fig. 6.1. Tipuri de subprograme

- În funcţie de locul în care sunt definite subprogramele *PL/SQL* pot fi:
  - locale
    - definite în partea declarativă a unui bloc *PL/SQL* sau a unui alt subprogram
  - independente
    - stocate în baza de date şi considerate drept obiecte ale acesteia
  - împachetate
    - definite într-un pachet *PL/SQL*

❖ Subprogramele împachetate sunt subprograme stocate?

❖ Un subprogram local, declarat şi apelat într-un bloc anonim, este temporar sau permanent? Poate fi apelat din alte aplicaţii?

❖ Un subprogram stocat este temporar sau permanent? Poate fi apelat din alte aplicaţii?

- De câte ori este compilat un subprogram local?
- De câte ori este compilat un subprogram stocat?

Avantajele utilizării subprogramelor stocate:

- codul este uşor de întreţinut
  - modificările necesare îmbunătăţirii mai multor aplicaţii trebuie realizate o singură dată
  - se minimizează timpul necesar testării
- codul este reutilizabil
  - după ce au fost compilate şi validate, subprogramele pot fi reutilizate în oricât de multe aplicaţii
- asigură securitatea datelor
  - acordând privilegii asupra subprogramelor, se poate permite accesul indirect asupra obiectelor bazei de date
- asigură integritatea datelor
  - se pot grupa mai multe acţiuni înrudite care vor fi executate împreună sau niciuna
- îmbunătăţesc performanţa
  - atunci când este creat un subprogram independent, în dicţionarul datelor este depus atât textul sursă, cât şi forma compilată (*p-code*)
  - dacă subprogramul este apelat, *p-code*-ul este citit de pe disc, este depus în *shared pool* şi poate fi utilizat de mai mulţi utilizatori
  - *p-code*-ul va părăsi *shared pool* conform algoritmului *LRU* (*least recently used*)

- Atunci când este apelat un subprogram stocat, *server*-ul *Oracle* parcurge etapele:
  - Verifică dacă utilizatorul are privilegiul de execuţie asupra subprogramului.
  - Verifică dacă *p-code*-ul subprogramului este în *shared pool*. Dacă este prezent va fi executat, altfel va fi încărcat de pe disc în *shared pool*.
  - Verifică dacă starea subprogramului este *VALID* sau *INVALID*. Starea unui subprogram este *INVALID*, fie pentru că au fost detectate erori la compilarea acestuia, fie pentru că structura unui obiect s-a schimbat de când subprogramul a fost executat ultima oară. Dacă starea subprogramului este *INVALID*, atunci este recompilat automat. Dacă nu a fost detectată nicio eroare, atunci va fi executată noua versiune a subprogramului.

- Dacă subprogramul aparţine unui pachet atunci toate subprogramele pachetului sunt de asemenea încărcate în *shared pool* (dacă acestea nu erau deja acolo). Dacă pachetul este activat pentru prima oară într-o sesiune, atunci *server*-ul va executa blocul de iniţializare al pachetului.

6.1. Proceduri *PL/SQL*

- O procedură este un bloc *PL/SQL* cu nume care poate accepta parametrii.
- În general procedurile sunt utilizate pentru a realiza anumite acţiuni.
- Procedurile independente sunt compilate şi stocate în baza de date ca obiecte ale schemei. Tipul acestor obiecte este *procedure.*

6.1.1. Definirea unei proceduri

- Sintaxa

```
[CREATE [OR REPLACE]] PROCEDURE  nume_procedură
          [(parametru[, parametru]...)]
    [AUTHID {DEFINER | CURRENT_USER}]
    {IS | AS}
    [PRAGMA AUTONOMOUS_TRANSACTION;]
    [declaraţii locale]
BEGIN
    secţiune executabilă
 [EXCEPTION
    secţiune de gestiune a excepţiilor]
END [nume_procedură];
```

BLOC PL/SQL

- Clauza *CREATE* determină stocarea procedurii în baza de date.
- Clauza *OR REPLACE* are ca efect ştergerea procedurii cu numele specificat (dacă aceasta există deja) şi înlocuirea acesteia cu noua versiune. Dacă procedura cu numele specificat în comanda *CREATE* există şi se omite clauza *OR REPLACE*, atunci apare eroarea „*ORA-955*: *Name is already used by an existing object*”.
- Parametrii specificaţi au următoarea formă sintactică:

```
parametru IN tip_de_date {:= | DEFAULT} expresie
          | { OUT | IN OUT }[NOCOPY] tip_de_date
```

Doar parametrii de tip *IN* pot avea specificate valori implicite (*DEFAULT*).

- Opţiunea *NOCOPY* poate fi utilizată doar pentru parametrii de tip *OUT* sau *IN OUT*. Determină baza de date să transmită parametrii de tip *OUT* sau *IN OUT* prin referință (parametrii de tip *IN* sunt transmiși doar prin referință, iar parametrii de *OUT* sunt transmiși implicit prin valoare). Atunci când se transmite o valoare mare (de exemplu, o colecţie), această clauză poate îmbunătăţi în mod semnificativ performanţa.
- Tipul parametrilor poate fi specificat utilizând atributele *%TYPE*, *%ROWTYPE* sau un tip explicit fără dimensiune specificată.

- Clauza *AUTHID* specifică faptul că procedura stocată se execută cu drepturile proprietarului (implicit) sau ale utilizatorului curent. De asemenea, această clauză precizează dacă referinţele către obiecte sunt rezolvate în schema proprietarului procedurii sau a utilizatorului curent.
- Clauza *PRAGMA AUTONOMOUS_TRANSACTION* informează compilatorul *PL/SQL* că această procedură este autonomă (independentă). Tranzacţiile autonome permit suspendarea tranzacţiei principale, executarea unor instrucţiuni *SQL*, *commit*-ul sau *rollback*-ul acestor operaţii şi continuarea tranzacţiei principale.

Exemplul 6.1 – vezi explicaţii curs

```
CREATE OR REPLACE PROCEDURE proc_ex1
     (v_id produse.id_produs%TYPE, v_procent NUMBER)
AS
BEGIN
  UPDATE  produse
  SET     pret_unitar =
               pret_unitar + pret_unitar*v_procent
  WHERE   id_produs = v_id;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR (-20000,'Nu exista produsul');
END;
```

Exemplul 6.2 – vezi curs

- Dacă subprogramul conţine comenzi *LMD*, atunci execuţia acestuia va determina execuţia tuturor *trigger*-ilor definiţi pentru aceste operaţii.
- Dacă se doreşte evitarea declanşării acestora, atunci înainte de apelarea subprogramului *trigger*-ii trebuie dezactivaţi, urmând ca aceştia să fie reactivaţi după ce s-a terminat execuţia subprogramului.

Exemplul 6.3

```
ALTER TABLE produse DISABLE ALL TRIGGERS;
EXECUTE proc_ex2(37,0.1)
ALTER TABLE produse ENABLE ALL TRIGGERS;
```

6.1.2. Apelarea unei proceduri

- Procedurile stocate pot fi apelate:
  - din corpul altei proceduri sau al unui declanşator;
  - interactiv, de utilizator folosind un utilitar *Oracle* (de exemplu, *SQL*Plus*);
  - explicit, dintr-o aplicaţie (de exemplu, *Oracle Forms* sau un precompilator).
- Apelarea unei proceduri se realizează
  - în *SQL*Plus*, prin comanda:
    `EXECUTE nume_procedură [(lista_parametri_actuali)];`
  - în *PL/SQL*, prin apariţia numelui procedurii urmat de lista parametrilor actuali:
    `nume_procedură [(lista_parametri_actuali)];`

O procedură stocată poate fi invocată într-o comandă *SQL* (de exemplu, în comanda *SELECT*)?

- La apelarea unei proceduri, parametrii actuali pot fi definiţi specificându-i
  - explicit, prin nume;
  - prin poziţie .

Exerciţiu 6.4 **- temă**

Definiţi un bloc *PL/SQL* în care procedura *proc_ex2* este apelată pentru fiecare produs din categoria „Sisteme de operare” (nivel 5). Preţul acestor produse va fi micşorat cu 5%.

Exemplul 6.5 **- vezi curs**

6.1.3. Transferul parametrilor

- Parametrii formali ai unei proceduri pot fi:
  - parametri de intrare (*IN*);
  - parametri de ieşire (*OUT*);
  - de intrare/ieşire (*IN OUT*).
- Dacă nu este specificat tipul parametrului, atunci implicit este considerat *IN*.
- Parametrul formal de tip *IN*
  - Poate primi valori implicite în cadrul comenzii de declarare.
  - Este *read-only* şi deci valoarea sa nu poate fi modificată în corpul subprogramului.
  - Parametrul actual corespunzător poate fi literal, expresie, constantă sau variabilă iniţializată.
- Parametrul formal de tip *OUT*
  - Este neiniţializat şi prin urmare, are automat valoarea *null*.
  - În interiorul subprogramului, parametrilor cu opţiunea *OUT* sau *IN OUT* trebuie să li se asigneze o valoare explicită. Dacă nu se atribuie nicio valoare, atunci parametrul actual corespunzător va fi *null*.
  - Parametrul actual trebuie să fie o variabilă, nu poate fi o constantă sau o expresie.
- Dacă în procedură apare o excepţie, atunci valorile parametrilor formali cu opţiunile *OUT* sau *IN OUT* nu sunt copiate în valorile parametrilor actuali.
- Implicit, transmiterea parametrilor este:
  - prin referinţă, în cazul parametrilor *IN*;
  - prin valoare în cazul parametrilor *OUT* sau *IN OUT*.
    - Dacă din motive de performanţă se doreşte transmiterea prin referinţă şi în cazul parametrilor *IN OUT* sau *OUT*, atunci se poate utiliza opţiunea *NOCOPY*.
    - Dacă opţiunea *NOCOPY* este asociată unui parametru *IN*, atunci va genera o eroare la compilare, deoarece aceşti parametri se transmit de fiecare dată prin referinţă.

`Exemplul 6.6` **– vezi curs**

`Exemplul 6.7` **– vezi curs**

6.2. Funcţii *PL/SQL*

- O funcţie *PL/SQL* este un bloc *PL/SQL* cu nume care trebuie să întoarcă un rezultat (o singură valoare).
- Funcţiile independente sunt compilate şi stocate în baza de date ca obiecte ale schemei. Tipul acestor obiecte este *function*.

O funcţie stocată poate fi invocată într-o comandă *SQL* (de exemplu, în comanda *SELECT*)?

6.2.1. Definirea unei funcţii

- Sintaxa

```
[CREATE [OR REPLACE]]  FUNCTION nume_funcţie
          [(parametru[, parametru]...)]
 RETURN tip_de_date
    [AUTHID {DEFINER | CURRENT_USER}]
    [DETERMINISTIC]
   {IS | AS}
    [PRAGMA AUTONOMOUS_TRANSACTION;]
    [declaraţii locale]
 BEGIN
    secţiune executabilă
 [EXCEPTION
    secţiune de gestiune a excepţiilor]
 END [nume_funcţie];
```

BLOC PL/SQL

  - Clauza *RETURN* este utilizată în locul parametrilor de tip *OUT*.
    - O funcţie trebuie să conţină clauza *RETURN* în antet şi cel puţin o comandă *RETURN* în partea executabilă.
      - În interiorul funcţiei trebuie să apară *RETURN expresie*, unde *expresie* este valoarea rezultatului furnizat de funcţie.
      - Într-o funcţie pot să apară mai multe comenzi *RETURN*, dar numai una din acestea va fi executată, deoarece după ce valoarea este întoarsă, procesarea blocului încetează.

- Algoritmul din interiorul corpului subprogramului funcţie trebuie să asigure că toate traiectoriile sale conduc la comanda *RETURN*. Dacă o traiectorie a algoritmului trimite în partea de tratare a erorilor, atunci *handler*-ul acesteia trebuie să includă o comandă *RETURN*.
- O funcţie fără comanda *RETURN* va genera eroare la compilare.
- Comanda *RETURN* (fără o expresie asociată) poate să apară şi într-o procedură. În acest caz, ea va avea ca efect revenirea la comanda ce urmează instrucţiunii apelante.

o Opţiunea *tip_de_date* specifică tipul valorii întoarse de funcţie, tip care nu poate conţine specificaţii de mărime.
- Dacă totuşi sunt necesare aceste specificaţii se pot defini subtipuri, iar parametrii vor fi declaraţi de subtipul respectiv.

o Opţiunea *DETERMINISTIC* indică optimizorului că funcţia va întoarce acelaşi rezultat dacă sunt folosite aceleaşi argumente la apelarea sa. Dacă sunt realizate apeluri repetate ale funcţiei, având aceleaşi argumente, atunci optimizorul poate utiliza un rezultat obţinut anterior.

- O funcţie poate accepta unul sau mai mulţi parametri. Ca şi în cazul procedurilor, lista parametrilor este opţională. Dacă subprogramul nu are parametri, parantezele nu sunt necesare la declarare şi la apelare.

Funcţiile acceptă toate cele 3 tipuri de parametrii (*IN*, *OUT* sau *IN OUT*).
În comenzile *SQL* pot fi utilizate doar funcţii cu parametrii de tip *IN*.

Ce tipuri de proceduri pot fi transformate în funcţii?

`Exemplul 6.8` **`– vezi curs`**

6.2.2. Apelarea unei funcţii

- O funcţie independentă poate fi apelată în mai multe moduri, folosind sintaxa:

  ```
  nume_funcţie [(lista_parametri_actuali)]
  ```

  - într-o comandă *SQL*;

    Exemplul 6.9

    ```
    SELECT func_ex8(100,2007)
    FROM   DUAL;
    ```

  - în *SQL*PLUS* (apelarea funcţiei şi atribuirea valorii acesteia într-o variabilă de legătură);

    Exemplul 6.10

    ```
    VARIABLE rezultat NUMBER
    EXECUTE :rezultat := func_ex8(100,2007);
    PRINT rezultat
    ```

  - în *PL/SQL*;

    Exemplul 6.11

    ```
    BEGIN
      DBMS_OUTPUT.PUT_LINE(func_ex8(100,2007));
    END;
    ```

- La apelarea unei funcţii, parametrii actuali pot fi definiţi specificându-i
  - explicit, prin nume;
  - prin poziţie.

    Exemplul 6.12

    ```
    DECLARE
      n NUMBER := func_ex8(p_id=>100);
    BEGIN
      DBMS_OUTPUT.PUT_LINE(n);
    END;
    ```

Comanda *CALL*

- Permite apelarea subprogramelor *PL/SQL* stocate (independente sau incluse în pachete) şi a subprogramelor *Java*.
- Este o comandă *SQL* care nu este validă într-un bloc *PL/SQL*.
  - În *PL/SQL* poate fi utilizată doar dinamic, prin intermediul comenzii *EXECUTE IMMEDIATE*.

- Sintaxa:

```
CALL nume_subprogram([lista_parametri actuali])
        [INTO :variabila_host]
```

  - *nume_subprogram* reprezintă numele unui subprogram sau numele unei metode.
  - Clauza *INTO* este folosită numai pentru variabilele de ieşire ale unei funcţii.

Exemplul 6.13

```
SQL> VARIABLE rezultat NUMBER
SQL> CALL func_ex8(100,2007) INTO :rezultat;

Call completed.

SQL> PRINT rezultat

  REZULTAT
----------
       863
```

6.2.3. Utilizarea în expresii *SQL* a funcţiilor definite de utilizator

- O funcţie stocată poate fi referită într-o comandă *SQL* la fel ca orice funcţie standard furnizată de sistem, dar cu anumite restricţii.
- Funcţiile definite de utilizator pot fi apelate din orice expresie *SQL* în care pot fi folosite funcţii *SQL* standard.
- Funcţiile definite de utilizator pot să apară în:
  - clauza *SELECT* a comenzii *SELECT*;
  - clauzele *WHERE* şi *HAVING*;
  - clauzele *CONNECT BY*, *START WITH*, *ORDER BY* şi *GROUP BY*;
  - clauza *VALUES* a comenzii *INSERT*;
  - clauza *SET* a comenzii *UPDATE*.

Exemplul 6.14 – vezi curs

Funcţiile ce pot fi utilizate în comenzi *SQL* trebuie:

- ❖ să aibă numai parametrii de tip *IN*, al căror tip de date este un tip valid *SQL*;
- ❖ să întoarcă o valoare al cărei tip să fie un tip valid *SQL*, cu dimensiunea maximă admisă în *SQL*.

Restricţii de utilizare a funcţiilor în comenzi *SQL*:

- funcţiile invocate într-o comandă *SELECT* nu pot conţine comenzi *LMD*;
- funcţiile invocate într-o comandă *UPDATE* sau *DELETE* asupra unei tabele *T* nu pot utiliza comenzi *SELECT* sau *LMD* care referă aceeaşi tabelă *T* (*table mutating*);
- nu pot termina tranzacţii (nu pot utiliza comenzile *COMMIT* sau *ROLLBACK*);
- nu pot utiliza comenzi *LDD* (de exemplu, *CREATE TABLE*) sau *LCD* (de exemplu, *ALTER SESSION*), deoarece acestea realizează *COMMIT* automat;
- nu pot să apară în clauza *CHECK* a unei comenzi *CREATE/ALTER TABLE*;
- nu pot fi folosite pentru a specifica o valoare implicită a unei coloane în cadrul unei comenzi *CREATE/ALTER TABLE*;
- nu pot utiliza subprograme care nu respectă restricţiile enumerate anterior.

6.3. Recompilarea subprogramelor *PL/SQL*

- Pentru a recompila subprogramele independente invalide se utilizează comanda

```
ALTER {FUNCTION | PROCEDURE} nume_subprogram COMPILE;
```

- Recompilarea explicită elimină recompilarea implicită la *run-time* şi previne apariţia erorilor de compilare la acel moment.

6.4. Ştergerea subprogramelor *PL/SQL*

- Pentru a şterge un subprogram independent se utilizează comanda:

```
DROP {FUNCTION | PROCEDURE} nume_subprogram;
```

6.5. Subprograme *overload*

- Subprogramele *overload* (supraîncărcate) au aceleaşi nume, dar diferă prin lista parametrilor.
  - *Exemplu*: funcţia predefinită *TO_CHAR*.
- În cazul unui apel, compilatorul compară parametri actuali cu listele parametrilor formali ale subprogramelor *overload* şi execută modulul corespunzător.

- Toate subprogramele *overload* trebuie definite în acelaşi bloc *PL/SQL* (bloc anonim sau pachet).
- Subprogramele *overload* pot să apară în programele *PL/SQL*:
  - în secţiunea declarativă a unui bloc;
  - în interiorul unui pachet.
- Subprogramele independente nu pot fi *overload*.
- Două subprograme *overload* trebuie să difere, cel puţin, prin tipul unuia dintre parametri. Două subprograme nu pot fi *overload* dacă parametri lor formali diferă numai prin subtipurile lor şi dacă aceste subtipuri se bazează pe acelaşi tip de date.
- Pentru ca două subprograme să fie *overload* nu este suficient ca:
  - lista parametrilor să difere numai prin numele parametrilor formali;
  - lista parametrilor să difere numai prin tipul acestora (*IN*, *OUT*, *IN OUT*); *PL/SQL* nu poate face diferenţe (la apelare) între tipurile *IN* sau *OUT*;
  - să difere doar prin tipul datei returnate (tipul datei specificate în clauza *RETURN* a unei funcţii).

Următoarele subprograme nu pot fi *overload*:

a)
```
FUNCTION alfa(v POSITIVE)…;
FUNCTION alfa(v PLS_INTEGER)…;
```
b)
```
FUNCTION alfa(x NUMBER)…;
FUNCTION alfa(y NUMBER)…;
```
c)
```
PROCEDURE beta(v IN VARCHAR2)…;
PROCEDURE beta(v OUT VARCHAR2)…;
```

`Exemplul 6.15` **– vezi curs**

6.6. Recursivitate

- Un subprogram recursiv se apelează pe el însuşi.
- Fiecare invocare recursivă a subprogramului creează câte o instanță pentru fiecare componentă declarată în subprogram și pentru fiecare comandă *SQL* executată de subprogram. Dacă apelul este în interiorul unui cursor *FOR* sau între comenzile *OPEN* şi *CLOSE*, atunci la fiecare apel este deschis alt cursor. Subprogramul poate determina depăşirea limitei impuse de parametrul *OPEN_CURSORS*.

`Exemplul 6.16` **– vezi curs**

6.7. Declaraţii *forward*

- Două subprograme sunt reciproc recursive dacă ele se apelează unul pe altul direct sau indirect.
- Declaraţiile *forward* permit:
  - definirea subprogramelor reciproc recursive;
  - declararea subprogramelor în ordine alfabetică sau într-o anumită ordine logică (în blocuri *PL/SQL* sau pachete).
- În *PL/SQL*, un identificator trebuie declarat înainte de a-l folosi. De asemenea, un subprogram trebuie declarat înainte de a-l apela.

Exemplul 6.17

```
DECLARE
 PROCEDURE alfa IS
 BEGIN
   beta('apel beta din alfa');    -- apel incorect
 END;
 PROCEDURE beta (x VARCHAR2) IS
 BEGIN
   DBMS_OUTPUT.PUT_LINE(x);
 END;
BEGIN
 alfa;
END;
```

- Eroarea apare deoarece procedura *beta* este apelată înainte de a fi declarată. Problema se poate rezolva simplu, inversând ordinea celor două proceduri. Această soluţie nu este eficientă întotdeauna.

```
DECLARE
 PROCEDURE beta (x VARCHAR2) IS
 BEGIN
   DBMS_OUTPUT.PUT_LINE(x);
 END;
 PROCEDURE alfa IS
 BEGIN
   beta('apel beta din alfa');    -- apel corect
 END;
BEGIN
 alfa;
END;
```

- *PL/SQL* permite un tip special de declarare a unui subprogram numit *forward*. Acesta constă dintr-o specificare de subprogram terminată prin ";".

```
DECLARE
 PROCEDURE beta (x VARCHAR2);  -- declaratie forward
 PROCEDURE alfa IS
 BEGIN
   beta('apel beta din alfa'); -- apel corect
 END;
 PROCEDURE beta (x VARCHAR2) IS
 BEGIN
   DBMS_OUTPUT.PUT_LINE(x);
 END;
BEGIN
 alfa;
END;
```

- Lista parametrilor formali din declaraţia *forward* trebuie să fie identică cu cea corespunzătoare corpului subprogramului. Corpul subprogramului poate apărea oriunde după declaraţia sa *forward*, dar trebuie să rămână în aceeaşi unitate de program.

6.8. Informaţii referitoare la subprograme

- Informaţiile referitoare la subprogramele *PL/SQL* şi modul de acces la acestea:
  - informaţii generale - vizualizarea *USER_OBJECTS*;
  - codul sursă - vizualizarea *USER_SOURCE*;
  - tipul parametrilor (*IN*, *OUT*, *IN OUT*) - comanda *DESCRIBE* din *SQL*Plus*;
  - erorile la compilare - vizualizarea *USER_ERRORS* sau comanda *SHOW ERRORS* din *SQL*Plus*.

Vizualizarea *USER_OBJECTS*

- Conţine informaţii generale despre toate obiectele bazei de date, în particular şi despre subprogramele stocate.
- Vizualizarea *USER_OBJECTS* conţine informaţii despre:
  - *OBJECT_NAME* – numele obiectului;

- *OBJECT_TYPE* – tipul obiectului (*PROCEDURE*, *FUNCTION* etc.);
- *OBJECT_ID* – identificator intern al obiectului;
- *CREATED* – data când obiectul a fost creat;
- *LAST_DDL_TIME* – data ultimei modificări a obiectului;
- *TIMESTAMP* – data şi momentul ultimei recompilări;
- *STATUS* – starea obiectului (*VALID* sau *INVALID*) etc.

- Pentru a verifica dacă recompilarea explicită (*ALTER*) sau implicită a avut succes se poate consulta starea subprogramelor utilizând *USER_OBJECTS*.
  - Orice obiect are o stare (*status*) sesizată în dicţionarul datelor, care poate fi:
    - *VALID* - obiectul a fost compilat şi poate fi folosit când este referit;
    - *INVALID* - obiectul trebuie compilat înainte de a fi folosit.

Exemplul 6.18

```
SELECT    OBJECT_ID, OBJECT_NAME, OBJECT_TYPE, STATUS
FROM      USER_OBJECTS
WHERE     OBJECT_TYPE IN ('PROCEDURE','FUNCTION');
```

Exemplul 6.19

```
CREATE OR REPLACE PROCEDURE proc_ex19 IS
BEGIN
  FOR i IN (SELECT  OBJECT_TYPE, OBJECT_NAME
            FROM    USER_OBJECTS
            WHERE   STATUS = 'INVALID'
            AND     OBJECT_TYPE IN ('PROCEDURE',
                      'FUNCTION')) LOOP
     --recompileaza toate subprogramele invalide
     --din schema curenta
     DBMS_DDL.ALTER_COMPILE(i.OBJECT_TYPE, USER,
                            i.OBJECT_NAME);
  END LOOP;
END;
```

- Dacă se recompilează un obiect *PL/SQL*, atunci *server*-ul va recompila orice obiect invalid de care depinde.
- Dacă recompilarea automată implicită a procedurilor locale dependente are probleme, atunci starea obiectului va rămâne *INVALID* şi *server*-ul *Oracle* semnalează eroare.
  - Este preferabil ca recompilarea să fie manuală (recompilare explicită utilizând comanda *ALTER* (*PROCEDURE*, *FUNCTION*, *TRIGGER*, *PACKAGE*) cu opţiunea *COMPILE*.

- Este necesar ca recompilarea să se facă cât mai repede, după definirea unei schimbări referitoare la obiectele bazei.

Vizualizarea *USER_SOURCE*

- După ce subprogramul a fost creat, codul sursă al acestuia poate fi obţinut consultând vizualizarea *USER_SOURCE* din dicţionarul datelor, care are următoarele câmpuri:
  - *NAME* – numele obiectului;
  - *TYPE* – tipul obiectului;
  - *LINE* – numărul liniei din codul sursă;
  - *TEXT* – textul liniilor codului sursă.

Exemplul 6.20

```
SELECT    TEXT
FROM      USER_SOURCE
WHERE     NAME = 'FIBONACCI'
ORDER BY  LINE;
```

6.9. Dependenţa subprogramelor

- Atunci când un subprogram este compilat, Atunci când un subprogram este compilat, în dicţionarul datelor se vor înregistra informații despre toate obiectele referite.
  - Subprogramul este dependent de aceste obiecte.
- Un subprogram care are erori la compilare este marcat ca "*invalid*" în dicţionarul datelor.
  - Un subprogram stocat poate deveni, de asemenea, invalid dacă o operaţie *LDD* este executată asupra unui obiect de care depinde.
- Dacă se modifică definiţia unui obiect referit, obiectul dependent poate (sau nu) să continue să funcţioneze normal.
- Există două tipuri de dependenţe:
  - dependenţă directă
    - obiectul dependent (de exemplu, *procedure* sau *function*) face referinţă direct la un *table*, *view*, *sequence*, *procedure* sau *function*
  - dependenţă indirectă
  - obiectul dependent (de exemplu, *procedure* sau *function*) face referinţă indirect la un *table*, *view*, *sequence*, *procedure*, *function* prin intermediul unui *view*, *procedure* sau *function*

- În cazul dependenţelor locale, atunci când un obiect referit este modificat, obiectele dependente sunt invalidate. La următorul apel al obiectului invalidat, acesta va fi recompilat automat de către *server*-ul *Oracle*.
- În cazul dependenţelor la distanţă, procedurile stocate local şi toate obiectele dependente vor fi invalidate. Acestea nu vor fi recompilate automat la următorul apel.
- Vizualizarea *USER_DEPENDENCIES* oferă informaţii despre obiectele referite de un obiect dependent.

Exemplul 6.21

```
SELECT NAME, TYPE, REFERENCED_NAME, REFERENCED_TYPE
FROM   USER_DEPENDENCIES
WHERE  REFERENCED_TYPE IN
       ('TYPE','TABLE','PROCEDURE','FUNCTION','VIEW')
AND    NAME NOT LIKE 'BIN%'
ORDER BY 1;
```

- Dependenţele indirecte pot fi afişate utilizând vizualizările *DEPTREE* şi *IDEPTREE*.
  - Vizualizarea *DEPTREE* afişează o reprezentare a tuturor obiectelor dependente (direct sau indirect).
  - Vizualizarea *IDEPTREE* afişează sub forma unui arbore aceleaşi informaţii.
- Pentru a utiliza aceste vizualizări furnizate de sistemul *Oracle* trebuie:
  1. conectare ca administrator;
  2. executat scriptul *UTLDTREE*;
  3. executată procedura *DEPTREE_FILL* (are trei argumente: tipul obiectului referit, schema obiectului referit, numele obiectului referit).

Exemplul 6.22

```
--conectare ca administrator
EXECUTE DEPTREE_FILL ('TABLE', 'CURS_PLSQL', 'FACTURI');

SELECT NESTED_LEVEL, TYPE, NAME
FROM   DEPTREE
ORDER  BY SEQ#;

SELECT  *
FROM    IDEPTREE;
```

- Dependenţele la distanţă pot fi manipulate folosind modelul *timestamp* (implicit) sau modelul *signature*.
- Ori de câte ori o unitate *PL/SQL* este modificată (creată sau recompilată) este înregistrat momentul de timp la care are loc modificarea (*timestamp*). Acesta poate fi observat interogând vizualizarea *USER_OBJECTS*, câmpul *LAST_DDL_TIME*. Modelul *timestamp* realizează compararea momentelor ultimei modificări a celor două obiecte analizate. Dacă un obiect (referit) are momentul ultimei modificări mai recent decât cel al obiectului dependent, atunci obiectul dependent va fi recompilat.
- Modelul *signature* determină momentul la care obiectele bazei distante trebuie recompilate. Atunci când este creată o unitate *PL/SQL*, o signatură este depusă în dicţionarul datelor, alături de *p-code*. Aceasta conţine numele blocului *PLSQL* (*PROCEDURE*, *FUNCTION*, *PACKAGE*), tipurile parametrilor, ordinea parametrilor, numărul acestora şi modul de transmitere (*IN*, *OUT*, *IN OUT*)), tipul de date întors de funcție. Dacă signatura nu este modificată, atunci execuţia continuă (fără a fi necesară recompilarea).

Recompilarea procedurilor şi funcţiilor dependente este fără succes dacă:

- ❖ obiectul referit este şters (*DROP*) sau redenumit (*RENAME*);
- ❖ tipul coloanei referite este schimbat;
- ❖ coloana referită este ştearsă;
- ❖ vizualizarea referită este înlocuită printr-o vizualizare având alte coloane;
- ❖ lista parametrilor unei proceduri referite este modificată.

Recompilarea procedurilor şi funcţiilor dependente este cu succes dacă:

- ❖ tabela referită are coloane noi;
- ❖ nicio coloană nou definită nu are restricția *NOT NULL*;
- ❖ tipul coloanelor referite nu s-a schimbat;
- ❖ tabela privată este eliminată, dar există o tabelă publică cu același nume şi structură;
- ❖ comenzile INSERT conţin efectiv lista coloanelor;
- ❖ un subprogram referit a fost modificat şi recompilat cu succes.

Cum pot fi minimizate erorile datorate dependenţelor?

- ❖ utilizând comenzi *SELECT* cu opţiunea *;
- ❖ incluzând lista coloanelor în cadrul comenzii *INSERT*;
- ❖ declarând variabile cu atributul *%TYPE*;
- ❖ declarând înregistrări cu atributul *%ROWTYPE.*

- Dacă procedura depinde de un obiect local, atunci se face recompilare automată la prima reexecuţie.
- Dacă procedura depinde de o procedură distantă, atunci se face recompilare automată, dar la a doua reexecuţie. Este preferabilă o recompilare manuală pentru prima reexecuţie sau implementarea unei strategii de reinvocare a ei (a doua oară).
- Dacă procedura depinde de un obiect distant, dar care nu este procedură, atunci nu se face recompilare automată.

6.10. Rutine externe

- *PL/SQL* a fost special conceput pentru *Oracle* şi este specializat pentru procesarea tranzacţiilor *SQL*.
- Totuşi, într-o aplicaţie complexă pot să apară cerinţe şi funcţionalităţi care sunt mai eficient de implementat în *C*, *Java* sau alt limbaj de programare. Dacă aplicaţia trebuie să efectueze anumite acţiuni care nu pot fi implementate optim utilizând *PL/SQL*, atunci este preferabil să fie utilizate alte limbaje care realizează performant acţiunile respective. În acest caz este necesară comunicarea între diferite module ale aplicaţiei care sunt scrise în limbaje diferite.
- Rutinele externe:
  - sunt subprograme scrise într-un limbaj diferit de *PL/SQL*;
  - sunt apelabile dintr-un program *PL/SQL*.
- *PL/SQL* extinde funcţionalitatea *server*-ului *Oracle*, furnizând o interfaţă pentru apelarea rutinelor externe. Orice bloc *PL/SQL* executat pe *server* sau pe *client* poate apela o rutină externă.
- Pentru a marca apelarea unei rutine externe în programul *PL/SQL* este definit un punct de intrare (*wrapper*) care direcţionează spre codul extern (program *PL/SQL* → *wrapper* → cod extern). O clauză specială (*AS EXTERNAL*) este utilizată (în cadrul comenzii *CREATE OR REPLACE PROCEDURE*) pentru crearea unui *wrapper*. De fapt, clauza conţine informaţii referitoare la numele bibliotecii în care se găseşte subprogramul extern (clauza *LIBRARY*), numele rutinei externe (clauza *NAME*) şi corespondenţa (*limbaj* <–> *PL/SQL*) între tipurile de date (clauza *PARAMETERS*). Ultimele versiuni renunţă la clauza *AS EXTERNAL*.

- Rutinele externe (scrise în *C*) sunt compilate, apoi depuse într-o bibliotecă dinamică (*DLL – dynamic link library*) şi sunt încărcate doar când este necesar acest lucru. Dacă se invocă o rutină externă scrisă în *C*, trebuie setată conexiunea spre această rutină. Un proces numit *extproc* este declanşat automat de către *server*. La rândul său, procesul *extproc* va încărca biblioteca identificată prin clauza *LIBRARY* şi va apela rutina respectivă.
- *Oracle8i* permite utilizarea de rutine externe scrise în *Java*. De asemenea, prin utilizarea clauzei *AS LANGUAGE*, un *wrapper* poate include specificaţii de apelare. De fapt, aceste specificaţii permit apelarea rutinelor externe scrise în orice limbaj. De exemplu, o procedură scrisă într-un limbaj diferit de *C* sau *Java* poate fi utilizată în *SQL* sau *PL/SQL* dacă procedura respectivă este apelabilă din *C*. În felul acesta, biblioteci standard scrise în alte limbaje de programare pot fi apelate din programe *PL/SQL*.
- Procedura *PL/SQL* executată pe *server*-ul *Oracle* poate apela o rutină externă scrisă în *C* care este depusă într-o bibliotecă partajată.
- Procedura *C* se execută într-un spaţiu adresă diferit de cel al *server*-ului *Oracle*, în timp ce unităţile *PL/SQL* şi metodele *Java* se execută în spaţiul de adresă al *server*-ului. *JVM* (*Java Virtual Machine*) de pe pe *server* va executa metoda *Java* direct, fără a fi necesar procesul *extproc*.
- Maniera de a încărca depinde de limbajul în care este scrisă rutina (*C* sau *Java*).
  - Pentru a apela rutine externe *C*, *server*-ul trebuie să cunoască poziţionarea bibliotecii dinamice *DLL*. Acest lucru este furnizat de *alias*-ul bibliotecii din clauza *AS LANGUAGE*.
  - Pentru apelarea unei rutine externe *Java* se va încărca clasa *Java* în baza de date. Este necesară doar crearea unui *wrappe*r care direcţionează spre codul extern. Spre deosebire de rutinele externe *C*, nu este necesară nici biblioteca şi nici setarea conexiunii spre rutina externă.
- Clauza *LANGUAGE* din cadrul comenzii de creare a unui subprogram, specifică limbajul în care este scrisă rutina (procedură externă *C* sau metodă *Java*) şi are următoarea formă:

  ```
  {IS | AS} LANGUAGE {C | JAVA}
  ```

- Pentru o procedură *C* sunt date informaţii referitoare la numele acesteia (clauza *NAME*); *alias*-ul bibliotecii în care se găseşte (clauza *LIBRARY*); opţiuni referitoare la tipul, poziţia, lungimea, modul de transmitere (prin valoare sau prin referinţă) al

parametrilor (clauza *PARAMETERS*); posibilitatea ca rutina externă să acceseze informaţii despre parametri, excepţii, alocarea memoriei utilizator (clauza *WITH CONTEXT*).

```
LIBRARY nume_biblioteca [NAME nume_proc_c] [WITH CONTEXT]
 [PARAMETERS (parametru_extern [, parametru_extern …] ) ]
```

- Pentru o metodă *Java*, în clauză trebuie specificată doar signatura metodei (lista tipurilor parametrilor în ordinea apariţiei).

Exemplul 6.23

```
CREATE OR REPLACE FUNCTION calc (x IN REAL)
RETURN NUMBER
  AS LANGUAGE C
LIBRARY biblioteca
NAME C_CALC
PARAMETERS (x BY REFERENCES);

-- apelare din PL/SQL
BEGIN
  calc(100);
END;
```

- Apelarea rutinei externe poate să apară în:
  - blocuri anonime;
  - subprograme independente sau aparţinând unui pachet;
  - metode ale unui tip obiect;
  - declanşatori bază de date;
  - comenzi *SQL* care apelează funcţii (în acest caz trebuie utilizată *PRAGMA RESTRICT_REFERENCES*).
- De remarcat că o metodă *Java* poate fi apelată din orice bloc *PL/SQL*, subprogram sau pachet.
- *JDBC* (*Java Database Connectivity*), care reprezintă interfaţa *Java* standard pentru conectare la baze de date relaţionale şi *SQLJ* permit apelarea de blocuri *PL/SQL* din programe *Java*. *SQLJ* face posibilă incorporarea operaţiilor *SQL* în codul *Java*. Standardul *SQLJ* acoperă doar operaţii *SQL* statice. *Oracle9i SQLJ* include extensii pentru a suporta direct *SQL* dinamic.
- Altă modalitate de a încărca programe *Java* este folosirea în *SQL*Plus* a comenzii: *CREATE JAVA instrucţiune*.

Bibliografie

1. Connolly T.M., Begg C.E., Database Systems: *A Practical Approach to Design, Implementation and Management*, 5th edition, Pearson Education, 2005
2. Dollinger R., Andron L., *Baze de date și gestiunea tranzacţiilor*, Editura Albastră, Cluj-Napoca, 2004
3. Oracle and/or its affiliates, *Oracle Database Concepts*, 1993, 2025
4. Oracle and/or its affiliates, *Oracle Database Performance Tuning Guide*, 2013, 2025
5. Oracle and/or its affiliates, *Oracle Database SQL Language Reference*, 1996, 2025
6. Oracle and/or its affiliates, *Oracle Database PL/SQL Language Reference*, 1996, 2025
7. Oracle and/or its affiliates, *Oracle Database Administrator's Guide*, 2001, 2023
8. Oracle and/or its affiliates, *Pro*C/C++ Programmer's Guide*, 1996, 2019
9. Oracle University, *Oracle Database*: *PL/SQL Fundamentals*, *Student Guide*, 2009, 2025
10. Popescu I., Alecu A., Velcescu L., Florea (Mihai) G., *Programare avansată în Oracle9i*, Ed. Tehnică, 2004